Tic-Tac-Toe AI using Minimax Algorithm with Alpha–Beta Pruning

This project implements a Tic-Tac-Toe game in Python where a human player competes against an AI opponent. The AI uses the Minimax algorithm enhanced with Alpha–Beta pruning to play optimally, meaning it cannot be beaten.

1. Overview

The program allows the user to play as 'X' or 'O'. The AI analyzes all possible future moves and selects the optimal one. The game continues until there is a winner or the board is full, resulting in a draw.

2. Board Representation

The board is represented as a list of 9 elements, corresponding to positions 0–8:

```
0 | 1 | 2
3 | 4 | 5
6 | 7 | 8
```

3. Minimax Algorithm

The Minimax algorithm is a recursive decision-making process used in two-player games. It assumes both players play optimally. The AI maximizes the score, while the human minimizes it. Scores are assigned as follows:

- +1 = AI wins
- -1 = Human wins
- 0 = Draw

4. Example of Decision-Making

If the AI is 'O' and the board is partially filled, the AI simulates every possible move, assumes the human will counter with the best possible response, and continues recursively until the game ends. It then backtracks and chooses the move leading to the highest score.

5. Alpha–Beta Pruning

Alpha–Beta pruning optimizes Minimax by skipping unnecessary branches of the game tree. If a branch cannot influence the final decision, it is pruned. This reduces computation time significantly without changing the final result.

- Alpha (α): The best score that the maximizing player (AI) can guarantee.
- Beta (β): The best score that the minimizing player (human) can guarantee.

When $\beta \leq \alpha$, the search stops for that branch.

6. Example of Alpha–Beta Pruning

Consider a small decision tree:

```
     (AI)
    /    \
  (1)      (Human)
          /     \
        (3)      (2)
```

The AI aims to maximize the score, and the human aims to minimize it. Once the AI finds a winning branch (e.g., score 1), it will prune any branch where the opponent could force a lower score, saving time without affecting accuracy.

7. Code Components

- Board functions: Manage and display the 3x3 grid.
- Minimax with Alpha–Beta pruning: Core AI logic for decision-making.
- choose_ai_move(): Determines the AI's best move using Minimax results.
- Game loop: Handles turns between the human and the AI, displaying results.

8. Summary

This project demonstrates how Minimax with Alpha–Beta pruning ensures optimal play in Tic-Tac-Toe. The AI will never lose if both players play perfectly. This project showcases practical application of game theory and search algorithms in Python.